ARMY RESEARCH LABORATORY

Battlefield Communications Network Model (BATNET)

Aivars Celmiņš

ARL-MR-244

August 1995

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

NOTICES

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1995	Final, Apr 94 - Dec 94

4. TITLE AND SUBTITLE
Battlefield Communications Network Model (BATNET)

5. FUNDING NUMBERS
62783094

6. AUTHOR(S)
Aivars Celmiņš

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-SC-CC
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-MR-244

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Typical future battlefield communication networks may consist of a moderate number of mobile radio nodes that broadcast on a common low-bandwidth channel. One can assume that the nodes will have high computing power, while the communications channel is slow and possibly noisy. In such a network, control of access times is necessary to avoid a breakdown of communications by colliding messages during times of high network traffic. To experiment with different control strategies and test their efficiency, a computer model of such a battlefield communication network has been developed at the U.S. Army Research Laboratory (ARL). The present report is a description of the model. The network traffic is modeled at an abstract level that permits the testing of the performances of various network access strategies and the assessing of effects of planned communication hardware. The model is coded in standard FORTRAN 77. When executed on a Cray supercomputer, it provides statistics about network performance typically in less than one percent of real time, that is, the modeling of one hour congested traffic requires a fraction of one minute on the computer. The model can be ported to any platform that supports FORTRAN 77.

14. SUBJECT TERMS
battlefield communications, communication network model, distributed communications control, cooperative control

15. NUMBER OF PAGES
51

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK

Contents

INTENTIONALLY LEFT BLANK

List of Figures

INTENTIONALLY LEFT BLANK

1. INTRODUCTION.

This report describes a computer model of a battlefield communications network that consists of a moderate number of nodes operating on a single radio channel. The nodes in a battlefield network are not stationary and they may become inoperable randomly for unpredictable lengths of time. The communications channel may also contain noise that is random or introduced on purpose. If we disregard the noise then the main obstacles to communication in such a network are message collisions. Because all messages are broadcast on the same channel, message routing is not possible and collision can be reduced only by controlling the accesses to the network. A central control that would assign broadcasting times to each node is not practical because in order to carry out such a control intelligently the controller would need current information about the state of the network, such as, the number of nodes, the lengths of their message queues, etc. To collect such information some broadcasting time would be needed that otherwise would be available for messages, and the information might be outdated on arrival. An alternative approach is to install at each node an independent access controller that would listen to the network traffic and regulate its own access time in such a manner that the overall information throughput rate is enhanced. A goal of ongoing research at the U.S. Army Research Laboratory is to devise algorithms for such controllers, that is, to develop a distributed cooperative control for battlefield networks.

Experiments with existing battlefield communication networks have shown that the behavior of the network is non-linear and difficult to predict theoretically (Kaste, Brodeen, and Broome 1992). Therefore, any new control concepts should ultimately be tested in experiments. A problem with such experimental investigations as described by Kaste, Brodeen, and Broome (1992) is that they are time consuming, expensive, and are limited to the use of existing hardware. To experiment with new protocols and new hardware concepts a computer model of a battlefield network offers many advantages by allowing for computational experiments which are cheaper, faster, and more flexible. Detailed computer models of large networks are available commercially. However, a simple and flexible in-house model that operates at a high level of abstraction is better suited for the development and testing of control procedures in a battlefield environment because a battlefield network simulator need not have as many options as simulations of stationary networks. (For instance, not needed are all the parameters that define the multitude of paths and node characteristics in a stationary network.) In the present report such a simple battlefield communications network model called BATNET is described.

The model is mainly intended for experiments with congested networks, that is with situations where several nodes have message queues waiting to be transmitted.

Section 2 contains an outline of the model, Section 3 defines the input format for the model, Section 4 describes the output, Section 5 presents some examples of experiments with the model, and Section 6 contains conclusions about the usefulness of the model. The program is coded in Fortran 77 and the code is listed in the Appendix.

2. PROGRAM OUTLINE

2.1. Program Structure

The principal part of the computer model BATNET is a subroutine called EXCASE that executes a case of network traffic and monitors the network activities. The queue management and access control for each node are simulated by a subroutine NODUPD that is called from EXCASE. On each call, NODUPD selects a message from the message queue of a node (see Section 2.3) and determines a time for the broadcasting of the message (see Section 2.5). The input arguments for the subroutine consist of a reference time, identification of a node, and parameters describing the message queue of the node. The output arguments are the identification of the message that is to be broadcast and the intended broadcast time. The network traffic is modeled as follows. First, the network monitor EXCASE determines that the network is free at time t_{free} and repeatedly calls NODUPD with t_{free} and each node as arguments in turn. Each call produces an intended broadcast time for the respective node. The monitor compares the intended broadcast times and assigns network access to the node with the smallest intended broadcast time. It then checks whether a collision takes place (see Section 2.6), determines whether the message is received and acknowledged, and tags the message correspondingly (see Section 2.3). The monitor then advances the time to the next free time spot, typically to the time after the message has been sent and acknowledged, and repeats the procedure with calling NODUPD. The broadcast times and broadcast types are stored in a time-line file (see Section 4). The subroutine EXCASE returns control to the main program when all queues are empty. The main program then may stop or call EXCASE to start another experiment.

The BATNET model is designed for experiments with congested networks, that is, with networks where several nodes have non-empty message queues. Therefore, each "experiment" is ended when the message queues have been emptied. The model can be used, with minor modifications, also to simulate stable network traffic, but that is not the intended application.

The code contains several input and output routines and a number of routines for the computation of various statistics that characterize the performance of the net. In a real life implementation, such network statistics would be computed independently and continuously at each node, and enable the controllers at the nodes to determine network access modes based on these statistics.

2.2. Representation of Messages.

The purpose of the BATNET model is to test general techniques for the control of a battlefield network. In these tests, a message can be represented by the time in seconds during which the communication channel is occupied with the broadcasting of the message and the corresponding acknowledgment. (We are not interested in character transfer rates and message coding.) A message in BATNET can be thought as consisting of the following four parts: a head that contains the address and other information for the decoding of the message; the main body containing the actual information to be transmitted; a hold time during which the addressee has the opportunity to start broadcasting an acknowledgment; a tail that constitutes the acknowledgment of the message. The broadcast length of a message in BATNET is the sum of at least the first two parts. The length of the tail is zero if the addressee is silent, and the tail and hold time are both zero if the message was not successfully transmitted due to collision or network disturbance. (It is assumed that the listening nodes can recognize disturbed messages and do not wait for an acknowledgment if a message is garbled.) The lengths of the four parts for the communications protocol described in Kaste, Brodeen, and Broome (1992) are as follows:

Head:	0.627 s
Body:	typically 0.5 to 10 s
Hold time:	1.000 s
Tail:	0.787 s

These values are adjustable parameters in BATNET and can be changed to represent other transfer protocols and transfer rates. The salient characteristic of this message model is that a message has a minimum length (0.627 s) and that the length of the hold time and acknowledgment (1.787 s), if existent, occupies the channel immediately after the broadcast of a message.

2.3. Queue Management.

For the modeling of message traffic in a congested network the lengths of the messages are the only important message characteristics. For the management of message queues, however, other message characteristics are more important. (Message lengths may be important for the queue manager only if message splitting and combination are considered.) Such characteristics are the

submission time and the priority of the message. The submission time is the time at which the message is submitted to a node for broadcasting. The priority is a number between zero and ten and is assigned to the message by the source (author) of the message. By assigning high priority numbers to important messages the source should increase the probability that such messages will be broadcast first. When a message is submitted the node manager augments the submitted message length by the length of the head (0.627 s) to obtain the queued length of the message and enters into the message queue a set of the following three numbers: submission time [s], queued message length [s], and a priority number.

In the BATNET model, the queues are established by the subroutine NODINI that is called from the main program before calling the case execution subroutine EXCASE (see Section 2.4). The lists of messages are stored in one floating point array and one integer array. The floating point array has three indexes and is called **flcue**. The array is constructed as follows:

First index = identification n of the node;

Second index = identification of a floating point characteristic:

flcue $(n, 1, i)$ = message submission time t_{oi} [s],

flcue $(n, 2, i)$ = queued message length L_i [s],

flcue $(n, 3, i)$ = message priority p in the range from zero to ten,

flcue $(n, 4, i)$ = message weight w_i (see below),

flcue $(n, 5, i)$ = time of last broadcast of the message t_{bi} [s];

Third index = identification i of the message.

The integer array also has three indexes and is called **incue**. This array is constructed as follows:

First index = identification n of the node;

Second index = identification of an integer characteristic:

incue $(n, 1, i)$ = identification number of addressee ("0" = "world"),

incue $(n, 2, i)$ = number of tried broadcasts N,

incue $(n, 3, i)$ = acknowledgment code: 0 - not sent, 1 - acknowledged, 2 - sent but not acknowledged, 3 - collided,

incue $(n, 4, i)$ = activity indicator: 0 - active, 1 - dormant;

Third index = identification i of the message.

The activity indicator **incue**$(n, 4, i)$ allows to identify as dormant, for instance, those messages that have been repeatedly sent to a not-responding receiver or are inactivated because of age. The dormant messages may be later reactivated when network traffic is low.

The queues are managed by the subroutine NODUPD that is called from EXCASE with a reference time and a node identification number as arguments. On each call, NODUPD computes weights for all messages in the queue of the argument node and selects the message with the largest weight for broadcasting.

The weight depends on the following characteristics of the messages:

Message priority p,
Elapsed time since message submission $t - t_o$, and
Number of unsuccessful broadcasts N.

At the beginning of a network experiment, each node is supplied with a list of future messages (see Section 2.4). The message queue of a node at a reference time t consists of all such messages from the list that are submitted before the time t and have not been broadcast successfully. The node management subroutine NODUPD assigns a zero weight to messages with a submission time t_o larger than t and a positive weight to other unsent messages. The formula for the positive message weight that is used in the BATNET model is as follows.

$$w = (1 + p + N) \cdot \max \left\{ 0.01 \, , \, \exp\left[- \left(\frac{t - t_o - 600}{600} \right)^2 \right] \right\} . \qquad (1)$$

The formula assigns higher weights to messages with higher priority p and to messages that have been repeatedly broadcast but not acknowledged. The weight also increases with elapsed time $t - t_o$ up to ten minutes (600 s). After that, the message is assumed to become stale and its weight is gradually reduced. At about 31 minutes after submission the weight ceases to depend on submission time and is very small compared to recently submitted message weights. (At $t = t_o$ the value of the exponential function is 0.37, ten minutes after submission it reaches unity and 31 minutes after submission it drops to 0.01.) The "stale time" of 600 s is arbitrary and can be replaced by another number. Such changes have, however, only little effect on the performance of a congested network because the weight merely determines which message from the queue has the first access to the network.

In practical applications the queue management might be different and the weight formula (1) modified to serve specific needs. For instance, messages with repetitions N larger than a threshold might be deactivated under the assumption that the receiving node is temporarily out of action. At the same time, all other messages in the queue with the same addressee could be deactivated. Furthermore, the weight function could be made dependent on the time that has elapsed since the last unsuccessful broadcast so that deactivated messages periodically would receive higher weights.

2.4. Message Generation.

At the start of a network traffic experiment with BATNET, lists of future messages are established for all active nodes. These lists are made by the subroutine NODINI that is called from the main routine in turn for each node. On each call, the subroutine generates a list of future messages as will be described below. If the model is used for the simulation of congested networks

then the algorithm of message generation is not important, because the network traffic conditions depend only on the first messages in the queues. It does not matter whether the remaining messages entered the queues randomly, at regular time intervals, or by some other scheme. The message generation algorithm described below was chosen for NODINI because it generates a wide variety of queues with the help of only few and easily understood parameters.

The input to BATNET specifies for each node a message frequency and an average submitted message length. The subroutine NODINI, called from BATNET, uses this information to generate messages with random submission times and random lengths. Let f [Hz] be the frequency of messages, l [s] be their average length, and *rand* be random numbers from an uniform distribution in the range [0,1]. Then the time interval between the submissions of the messages i and $i-1$ is computed by

$$\Delta_i = rand \cdot 2 \,/\, f \quad [\,\mathrm{s}\,]\ . \tag{2}$$

The submission time of the message i is

$$t_{oi} = \sum_{k=1}^{i} \Delta_k \quad [\,\mathrm{s}\,]\ , \tag{3}$$

and its queued length is

$$l_i = 0.627 + rand \cdot 2 \cdot l \quad [\,\mathrm{s}\,]\ , \tag{4}$$

where 0.627 s is the length of the mandatory head of a message. (The random numbers *rand* in Eqs. (2) and (4) are, of course, not the same.) The addressee and the priority of each message are also assigned randomly. This information is stored in two lists (arrays) that contain the data for all nodes (see Section 2.3). The end of a message list generation by NODINI is determined by the end time of message generation (an input datum), i. e., when t_{oi} exceeds the specified end time. If the declared arrays in the program are not sufficiently large for the storage of all messages up to the end time then BATNET writes an error message in the file named **<input-summary>** and stops.

2.5. Network Access Management.

The access control that is modeled by BATNET is intended to work as follows. When a node (with a non-empty message queue) observes that the network is free at time t_{free}, it selects a message from its queue (see Section 2.3) and computes an intended broadcast time $t_s = t_{free} + \Delta$. The waiting interval Δ is randomly chosen from a network access *delay time interval D* that is equal for all nodes. The node then continues to monitor the network and, if the network is still free at t_s, starts broadcasting at that time. With this algorithm, the node with the smallest intended broadcast time will broadcast, while other nodes will find that the net is not free at their t_s, abstain from broadcasting, and

wait for the next free time. At that time, the competition among nodes for the smallest t_s starts again. If the delay time interval D is the same for all nodes then this algorithm assures equal access probability for all messages at the heads of message queues. To provide a larger probability of access for high priority messages, the global D may be reduced for such messages. This reduction of D is implemented by the subroutine NODUPD as follows.

Let t be the reference time (provided by EXCASE when it calls NODUPD; at this time the network is free) and let D [s] be the network access delay time interval, also provided by the calling program. (This parameter is presently constant and obtained from input. One of the goals of ongoing research is to find algorithms that control and vary D such that the information throughput is increased.) If the priority of the selected message is zero then NODUPD chooses a random number Δ from the interval $(0\ ,\ D)$ and computes the intended broadcast time by

$$t_s = t + \Delta \quad . \tag{5}$$

If the priority p of the message is positive then Δ is chosen from a smaller interval $(0\ ,\ D_{local})$ where

$$D_{local} = (1 - 0.09 \cdot p) \cdot D \quad . \tag{6}$$

With this algorithm, a message with the highest priority ($p = 10$) has in the average a ten times smaller Δ than a low priority message.

The calling program EXCASE compares the intended broadcast times of all nodes and assigns broadcasting to the node with the smallest t_s by tagging the corresponding message as sent (see Section 2.3).

2.6. Message Collisions.

If the access control described in Section 2.5 could be carried out with infinitesimal accuracy then the probability of message collisions would be zero. In reality, the accuracies of the intended broadcast times t_s are finite and in addition also delays in the network communications (propagation time and equipment-induced delays) can cause collisions. In particular, the following takes place at the node with the smallest intended broadcast time t_s and in the network:

The node determines that the channel is free.
The node starts broadcasting.
Other nodes recognize that a broadcasting takes place.

Let the time interval between t_s and the recognition by the other nodes be α. Then any other node that has an intended broadcast time between t_s and $t_s + \alpha$ will falsely determine that the channel is free, start broadcasting, and cause collisions of messages.

For the BATNET model the collision interval α is an input item. A reasonable value for α is about 0.5 s.

3. INPUT

The input for an experiment with BATNET is read from an input file. Figure 1 shows an example of such a file. The first line of the input file contains an alphanumeric identification. The second line is merely an explanation of the input. The contents of this and subsequent explanation lines are ignored by the program but the lines must be in the input file. Line 3 contains two numbers. These numbers and all other numerical input data are not formatted and the reading of the numbers is done in list-directed mode. The first number is the *collision interval* α, that is, the interval between message broadcast times that determines whether a collision takes place. Any message that has a broadcast time within this interval after the first broadcast message is assumed to be broadcast, too, and colliding with the first message (see Section 2.6). The second number is the *hold time,* that is, the time interval after a message during which the addressee should start an acknowledgment (see Section 2.2). Line 5 contains three parameters for the control of the network access *delay time interval D.* Presently only the first parameter is used and taken to be the global value of D. The other two numbers are dummy parameters and included in the input for future use.

Line 7 in Figure 1 indicates that the network has four nodes. After another explanation line, the input lists for the four nodes *message frequencies f* and *message lengths l.* These values are used by the subroutine NODINI to generate lists of messages (see Section 2.4). Because each input line for the frequencies and message lengths also contains the identification number of the node, the sequence of these data lines is arbitrary. If the number of nodes in n then the node description ends with Line $8+n$.

Line $8+n+2$ contains a *seed number* for the random number generator. The seed number is needed to enable repeated computations with identical random number sequences. Line $8+n+4$ contains the *end time* for message generation. The last Line $8+n+6$ contains the *monitoring time interval L.* This is the interval that is used by the subroutine MONITOR to calculate time-averaged network statistics, such as the percentage of time spent on broadcasting, on idling, and on transmitting collided messages. (See Section 4 and Figures 5, 9, and 10.)

The input is read by the subroutine READER from the file unit 21. Therefore, a file `<fort.21>` must be linked to the input file before the code is run, or the input file must be copied to `<fort.21>`.

```
Line 1.        Four different nodes. 940520.  Delay interval = 10 [s]
Line 2.        Collision interval [s], hold time [s]
Line 3.           .5000             1.0000
Line 4.        Message delay parameters (3)
Line 5.           10.00      1.00     1.00
Line 6.        Number of nodes in the network
Line 7.                4
Line 8.        Node No.  Messg. freq. [Hz]  Messg. length [s]
Line 8+1.         1          .030                 9.0
Line 8+2.         2          .070                 3.6
Line 8+3.         3          .130                 2.0
Line 8+4.         4          .187                 0.9
Line 8+n+1.    Seed for the random number generator
Line 8+n+2.        1188
Line 8+n+3.    End time of message generation [s]
Line 8+n+4.      600.0
Line 8+n+5.    Monitoring time [s]
Line 8+n+6.      180.0
```

Figure 1. Input file for Experiment No. 1.

Next, BATNET calls a subroutine WRITER that writes a formatted printable summary of the input, supplemented with computing date and time, into a file named `<input-summary>`. The format of that summary is the same as shown in Figure 1, so that this summary file can also be used as input for repeat calculations.

Next, the main program calls the subroutine NODINI to generate message lists (see Section 2.4). After the message generation, another output routine WRITE2 is called to supplement the file `<input-summary>` with some statistics about the message lists. At the end of the experiment, the main program adds to the file a line with the time that was needed in the experiment to clear all queues. The supplements to the input file are illustrated in Figure 2.

4. OUTPUT

BATNET output consists of a number of network activity records in separate files. One such file, the `<input-summary>`, was described in Section 3. Another output file `<time-line>` contains the time record of network activities. Figure 3 shows the beginning of the file from Experiment No. 1 that was run with the input shown in Figure 1. The complete file has in this case 707 lines. The file shows that the first successful broadcast was completed at $t = 4.26$ s and that the length of the broadcast message was 2.76 s. The first collided broadcast was finished at $t = 27.58$ s and lasted 2.28 s, etc.

```
Computing date 09/13/94.   Computing time 08:45:58.

   Summary of unsent messages at time=   .000

      First subm.   Last subm.  Total msg.  Total msg.  Average msg.
Node   time [s]      time [s]    number     length [s]   length [s]

  1      1.501       601.168       17        166.155      9.774
  2     20.786       600.271       42        166.766      3.971
  3     13.411       607.040       73        192.410      2.636
  4      7.150       604.418      116        184.523      1.591

             Total:               248        709.854

Total length including acknowledgments =  1069.039

All queues are cleared at  2179.54 [s]
```

Figure 2. Supplements to the input file of Experiment No. 1.

```
Computing date 09/13/94.   Computing time 08:45:58.
Four different nodes. 940520.  Delay interval = 10 [s]
 End time [s],  interval [s],  activity code
  1 = idle, 2 = transmission, 3 = collision
  4 = interference, 5 = not acknowledged
    1.5007     1.5007    1
    4.2649     2.7642    2
    7.1497     2.8848    1
   10.1393     2.9896    2
   13.4109     3.2717    1
   18.5910     5.1801    2
   20.5391     1.9481    1
   24.9973     4.4582    2
   25.2944      .2972    1
   27.5752     2.2807    3
   29.3285     1.7533    1
   33.0445     3.7160    2
   ....................
```

Figure 3. Output file <time-line> of Experiment No. 1.

The file <time-line> and the other output files to be described are in general too voluminous for reading by humans. They are intended as input files for graphical display or statistical analysis. Each output file has a head with a comprehensive explanation of its contents to facilitate coding of routines that read the data for statistical analyses of graphical display. We show here only excerpts of some output files as illustrations.

Four output files contain information about the lengths of message queues and waiting times at a series of sampling times. The information about the queue lengths is obtained as one of the output arguments of the subroutine NODUPD. Information about waiting times in each queue is compiled and written into the file `<cues-de>` by a subroutine named WDELAY. The names of these four output files are as follows.

`<cues-nr>` - numbers of unsent and repeated messages

`<cues-ti>` - queue lengths in terms of sums of queued message lengths

`<cues-we>` - queue lengths in terms of sums of message weights

`<cues-de>` - maximum and ensemble averages of waiting times

Figure 4 contains a part of the output file `<cues-ti>` from the Experiment No. 1. The sampling times in Column (1) are the times at the end of a broadcast activity (see the time line in Figure 3). The lengths of the queues are averages over the monitoring time interval L (=180 s) before the sampling time. At the beginning of the experiment, when $t < L$, the averaging is done over the time elapsed since the beginning of the experiment. For instance, for $t = 100.0380$ s at the Node No. 2 the average queue length during the time from zero to 100.0380 was 20.41 s. The figure shows how the queues increase as new messages are submitted and decrease again as the messages with largest weights are broadcast.

The output files with information about queue lengths permit a comparison of performances by different nodes. Individual nodes that listen to the network do not have access to this information. Information that is available to individual nodes is computed by the subroutine MONITOR that is called from EXCASE after each idle or broadcast interval with the reference time as argument. MONITOR computes average network usage statistics for a given time interval L immediately before the reference time. The averaging interval L is specified by input, it is the "Monitoring time" in Figure 1. In all presented examples we have $L = 180$ s. Experimentation with different values of L have shown that 180 s is appropriate if the typical message length is of the order of six seconds. A too small averaging interval makes the detection of trends of network usages difficult. A too large averaging interval suppresses the latest information. MONITOR computes two types of averages: a simple unweighted time average and a weighted average where the weight decreases linearly from the beginning to the end of the averaging interval. The difference between the weighted and unweighted averages can be used as a trend indicator of the averaged quantity. (A simple numerical derivative is not useful as a trend indicator because of the small oscillations in the averages as shown, for instance, in Figure 9.) MONITOR uses as the averaging interval either the input value of L (=180 s) or the time from the beginning of the experiment, whichever is less. The output by the subroutine MONITOR is stored in two output files called `<monitor-line>` and `<monitor2-numbers>`.

```
Computing date 09/13/94.   Computing time 08:45:58.
Four different nodes. 940520. Delay interval = 10 [s]
   180.0 [s] = the monitored time interval
   4 = number of nodes
  Time          Lengths [s] of information in queues
   .0000      .0000        .0000       .0000      .0000
  4.2649      .0000        .0000       .0000      .0000
 10.1393      .0000        .0000       .0000      .0000
 18.5910    16.2685        .0000      2.6712     3.0791
 24.9973    16.2685       1.6159      3.3053     5.4225
 27.5752    16.2685       1.6159      3.3053     6.4197
 33.0445    16.2685       7.2982      3.3053     5.3961
 40.8458    16.2685       1.6159      8.2129     8.1994
 43.8621    16.2685       1.6159      8.2129     8.7886
 61.9401      .0000       4.2726     10.7185    15.5826
 66.2095      .0000       6.4249      8.8659    17.5495
 69.8474      .0000       6.4249     10.1254    19.1287
 74.9873      .0000       6.4249     10.1254    18.6160
 79.3721      .0000      11.9840     10.1254    18.6160
 88.1773     7.2568      17.7552     11.2001    19.7031
 92.0413     7.2568      17.7552     11.2001    19.3107
100.0380     7.2568      20.4104     13.2358    21.9745
104.7375     7.2568      20.4104     13.2358    20.3953
 ........................................
```

Figure 4. Output file <cues-ti> of Experiment No. 1.

Figure 5 shows a part of the file <monitor-line> from the Experiment No. 1. The complete file has 1412 lines. The first column contains the end times of messages and idle periods, that is, the same time values as in Figure 3. The second column lists the average length of all not-colliding messages that were sent during the averaging time interval before the reference time in the first column. (Idle time and colliding messages are not included in this calculation. If a message overlaps the averaging interval then only that part is considered that is within L.) The next two columns list the average usage of network time by idling: Column (3) contains the relative time in percent of L and Column (4) contains the average length of idling intervals in seconds. The subsequent line contains in Columns (3) and (4) the same quantities computed with weighted averaging. The difference between the second and the first line can be used as a trend indicator of the data.

The remaining columns contain in the same format network usage data for the following categories: transmissions, collisions. interferences, and not acknowledged messages. For instance, at $t = 13.411$ s the average length of all messages was 2.877 s, idling had occupied 57.1% of network time, the average length of an idle interval was 2.55 s, transmissions used 42.9% of network time and the average length of transmissions was 2.88 s. The average message

```
Computing date 09/13/94.  Computing time 08:45:58.
Four different nodes. 940520.  Delay interval = 10 [s]
   180.0 [s] = the monitored time interval
 Column (1) = time; (2)  = active interval length [s]
 (3,4) relative length [%] and length [s] of idling
 (5,6) the same for transmission, (7,8) = collision
 (9,10)  = interference, (11,12) = not acknowledged
 Second line = the same with weighted calculation
   1.501    .000 100.0   1.50    .0    .00    .0    .00    .0    .00    .0    .00
                 100.0   1.50    .0    .00    .0    .00    .0    .00    .0    .00
   4.265  2.764   35.2   1.50  64.8   2.76    .0    .00    .0    .00    .0    .00
                  42.8   1.50  57.2   2.76    .0    .00    .0    .00    .0    .00
   7.150  2.764   61.3   2.19  38.7   2.76    .0    .00    .0    .00    .0    .00
                  58.8   2.04  41.2   2.76    .0    .00    .0    .00    .0    .00
  10.139  2.877   43.3   2.19  56.7   2.88    .0    .00    .0    .00    .0    .00
                  46.3   2.09  53.7   2.85    .0    .00    .0    .00    .0    .00
  13.411  2.877   57.1   2.55  42.9   2.88    .0    .00    .0    .00    .0    .00
                  55.3   2.40  44.7   2.86    .0    .00    .0    .00    .0    .00
  18.591  3.645   41.2   2.55  58.8   3.64    .0    .00    .0    .00    .0    .00
                  44.1   2.45  55.9   3.45    .0    .00    .0    .00    .0    .00
  20.539  3.645   46.8   2.40  53.2   3.64    .0    .00    .0    .00    .0    .00
                  47.5   2.37  52.5   3.47    .0    .00    .0    .00    .0    .00
.............................................
```

Figure 5. Output file <monitor-line> of Experiment No. 1.

lengths in Columns (2) and (6) are equal because during the monitoring time the network was used only for successful transmissions.

```
Computing date 09/13/94.  Computing time 08:45:58.
Four different nodes. 940520.  Delay interval = 10 [s]
   180.0 [s] = the monitored time interval
 Column (1) = time [s];
 (2) - (6) = relative numbers [%] of accesses during dtimon:
  2 = idle, 3 = transmission, 4 = collision
  5 = interference, 6 = not acknowledged
 Second line = the same with weighted calculation
   1.5007 100.000000    .000000    .000000    .000000    .000000
           75.000000    .000000    .000000    .000000    .000000
   4.2649 100.000000 100.000000    .000000    .000000    .000000
          137.762634 100.000000    .000000    .000000    .000000
.............................................
  33.3765 116.666664  83.333328  16.666668    .000000    .000000
          117.110291  86.220726  13.779278    .000000    .000000
  40.8458 100.000000  85.714287  14.285715    .000000    .000000
          104.784988  87.053612  12.946394    .000000    .000000
  41.1045 114.285721  85.714287  14.285715    .000000    .000000
          114.324318  87.042328  12.957665    .000000    .000000
.............................................
```

Figure 6. Output file <monitor2-numbers> of Experiment No. 1.

Figure 6 shows the output file **<monitor2-numbers>** that contains the relative number of accesses to the network during the monitored time interval. For instance, at $t = 33.3756$ s the network was accessed for successful message transmission in 83.3% of all accesses and for sending collided messages in 16.7% of all cases. The percentage is calculated in terms of message accesses without counting an idle interval as "access". Therefore, a formal calculation of the number of idling "accesses" yields in the discussed entry line the value of 116.7%. (In general this value should be about 100% because within the

monitoring time interval the number of idling intervals is always about equal to the total number of all accesses.) Every other line in the file shows the weighted averages of access numbers. As discussed above, the difference between weighted and unweighted values indicate the trend of the averaged quantity.

5. EXAMPLES

We present in this section results from experiments with the BATNET code. In the first experiment, we consider a network with four nodes that is defined by the input shown in Figure 1. The number of messages and their frequencies are different for different nodes, but the message lengths and frequency parameters were chosen such that the combined lengths of all messages are approximately equal for all nodes. This means that in this experiment the amount of information that is submitted to each node during the message generation time (about 600 s) is approximately the same while the average lengths of messages and frequencies of message submissions vary widely. The total queued length of submitted messages is 710 s, but one needs at least 1069 s of network time to transmit the messages and their corresponding acknowledgments (see Figure 2). In reality, the communication channel will be used for a substantially longer time because of the idling between messages and the time that is wasted with colliding messages. In this experiment, the clearing of all queues required 2179 s or about 36 minutes.

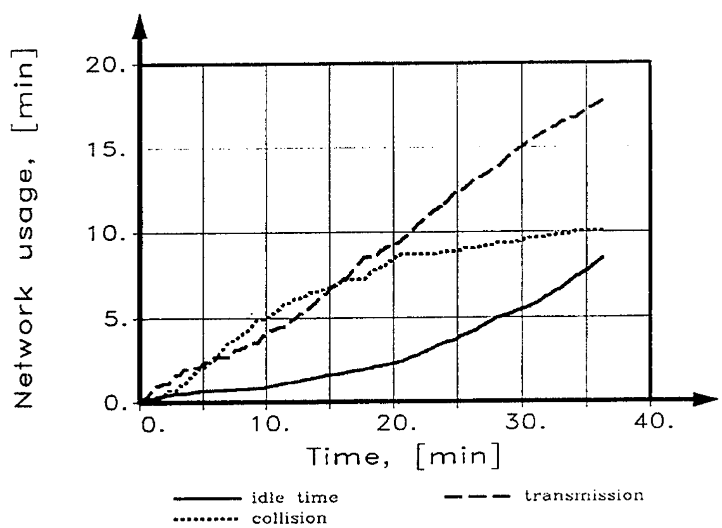

Figure 7. Cumulative network usage times in Experiment No. 1.

Figure 7 shows the network usage times in Experiment No. 1. During the 36 minutes that were needed to clear all queues, the network was idle for about 8 minutes and was transmitting colliding messages for about 10 minutes. The collision time can be easily reduced by increasing the network access delay time

interval D. To demonstrate the effect of changing D a second experiment was conducted for the same set of messages but with an access delay time interval D increased from 10 s to 20 s. The result of this "Experiment No. 2" is shown in Figure 8. The collision time is indeed reduced to about one quarter of the previous experiment but now the idle time has increased from 8 to 15 minutes. All queues are emptied in 35 minutes. The example shows that the network performance is quite sensitive to the access delay time interval. It also indicates that with a dynamic adjustment of the delay time interval D one might achieve better results than with a fixed D. If the communication traffic is light then a small value of D is advantageous, because it reduces the idle time. In a heavy traffic situation D must be made larger to reduce collisions.

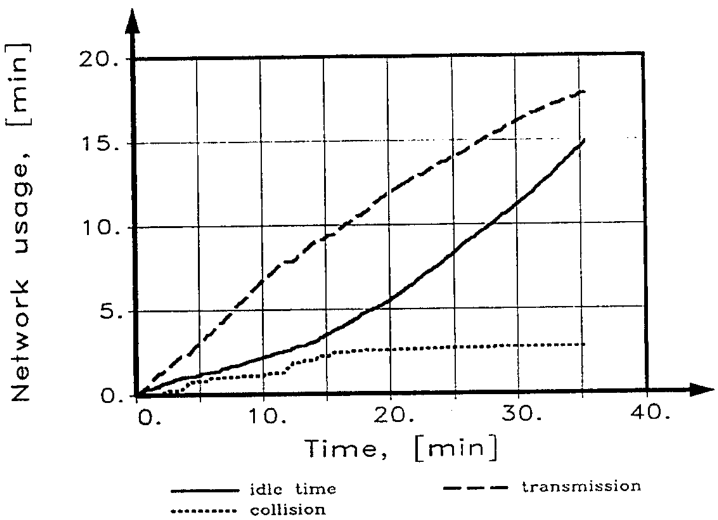

Figure 8. Cumulative network usage times in Experiment No. 2.

An example of network statistics that is available to a listening node is shown in Figure 9. It displays for the first experiment ($D = 10$ s) the network usage in percent for different usage categories during the 180 s monitoring interval. For instance, at $t = 5$ minutes we see that during the time between 2 and 5 minutes the network was occupied 50% with collisions, 40% with successful transmissions, and 10% with idling. A listening node with local control would notice the large amount of collision time and increase the value of D, for instance, starting at about $t = 5$. Towards the end of the experiment D could be decreased again because there the idling time becomes substantial.

Figure 10 shows the network usage statistics in Experiment No. 2 where the larger $D = 20$ s was used. We notice that the larger D indeed has reduced the many collision at the beginning of the experiment, but towards the end of the experiment the amount of idle time exceeds that of transmissions. The total time for the clearance of the queues could be reduced if a dynamic assignment of

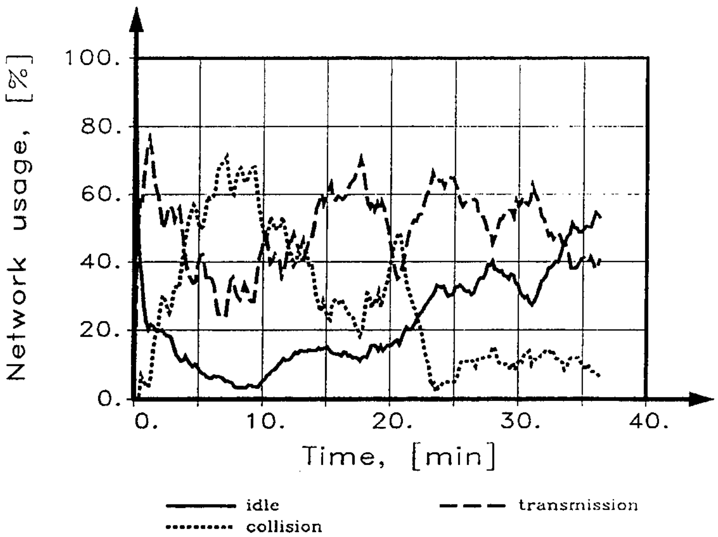

Figure 9. Average network usage in Experiment No. 1.

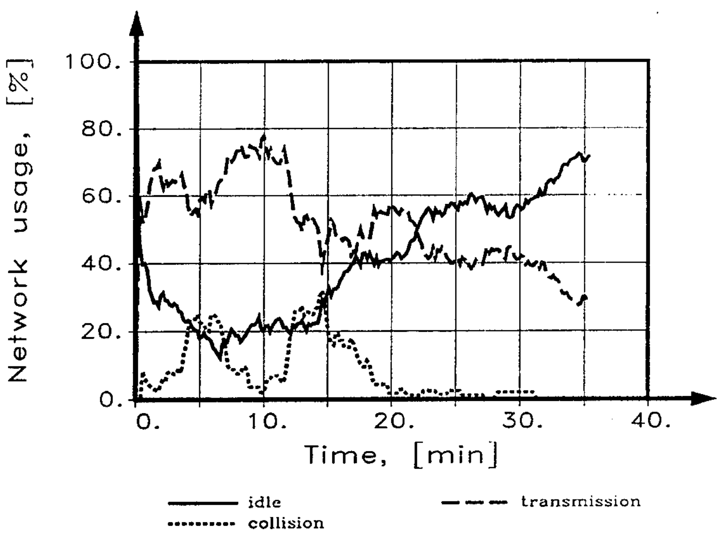

Figure 10. Average network usage in Experiment No. 2.

D were used to reduce idling during periods when the number of collisions is low.

Other useful statistics that can be obtained by listening to the network are, for instance, the message lengths, the number of accesses, and the number of active nodes. It remains to be seen how such information can be used for network control.

An interesting statistic that is not available to the listening nodes but is available in this model to the network monitor is the length of message queues.

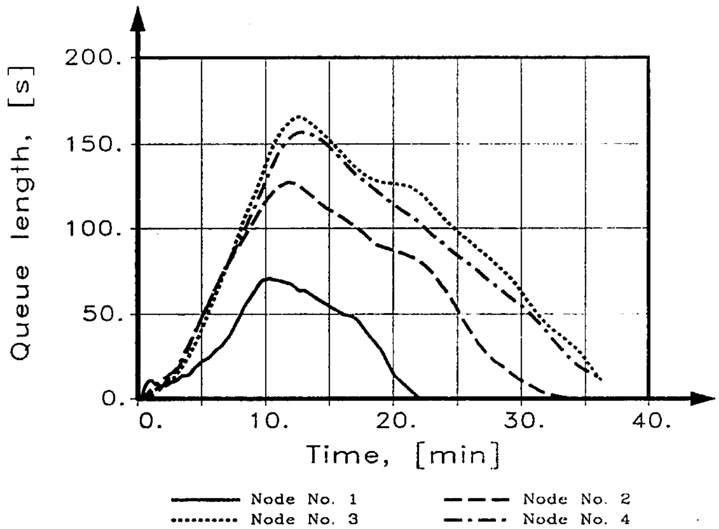

Figure 11. Average queue lengths in Experiment No. 1.

Figure 11 shows the average queue lengths in the Experiment No. 1 with four nodes and $D_1 = 10$ s. The lengths are expressed in seconds as sums of queued message lengths and averaged over 180 s. For instance, the figure shows that at $t = 15$ minutes the average queue length during the time from 12 to 15 minutes was 151 s at Node No. 4 and 51 s at Node No. 1. It is obvious from Figure 11 that the nodes with fewer and longer messages empty their queues sooner. This is not surprising since a node with many short messages uses more network access delay time and must compete more often with other nodes for network access. In a noiseless network (as in this experiment) it is, therefore, advantageous to combine messages in packets.

In the next two experiments the BATNET code was executed for a network with ten equal nodes, that is, for a network where the message frequencies and average message lengths are the same for all nodes. Figure 12 shows the input part of the file `<input-summary>` and Figure 13 shows the supplement part of the file. The average queued length of a message was about 4.6 s, and a total of 241 messages were submitted in 600 s. The numbers of messages per node are between 22 and 26. In the Experiment No. 3 we used a network access delay parameter $D = 20$ s and obtained network usage times as shown in Figure 14. It is obvious that the delay time parameter which was in general too large for the four-node network is too small for ten nodes: during the first 25 minutes the collision time is as large as the transmission time. To reduce the collisions we increased the delay time parameter to $D = 50$ s in the next Experiment No. 4. The corresponding network usage times are shown in Figure 15. A comparison of Figure 14 with Figure 15 shows that the increase of D has improved the network utilization. As in the four-node network, further improvements could

```
Ten equal nodes.  940520.    Delay interval = 20 [s]
 Collision interval [s], hold time [s]
    .5000   1.0000
 Message delay parameters (3)
    20.00      1.00      1.00
 Number of nodes in the network
       10
 Node No.  Messg. freq. [Hz] Messg. length [s]
   1           .040              4.0
   2           .040              4.0
   3           .040              4.0
   4           .040              4.0
   5           .040              4.0
   6           .040              4.0
   7           .040              4.0
   8           .040              4.0
   9           .040              4.0
  10           .040              4.0
 Seed for the random number generator
      1188
 End time of message generation [s]
    600.0
 Monitoring time [s]
    180.0
```

Figure 12. Input part in **<input-summary>** for Experiment No. 3.

be achieved by a dynamic assignment of the parameter. A comparison of the experiments involving four and ten nodes, respectively, again indicates that an optimal network control must be dynamic since the size of a battlefield network (the number of active nodes) varies with time and cannot be used to determine access control parameters in advance.

Computing date 09/13/94. Computing time 09:31:48.

Summary of unsent messages at time= .000

Node	First subm. time [s]	Last subm. time [s]	Total msg. number	Total msg. length [s].	Average msg. length [s]
1	1.126	604.328	23	107.188	4.660
2	19.050	629.708	25	109.951	4.398
3	3.123	619.942	22	88.075	4.003
4	17.230	604.079	23	105.828	4.601
5	10.633	637.609	24	118.828	4.951
6	27.055	609.393	26	115.484	4.442
7	19.754	624.656	26	139.010	5.347
8	48.742	614.667	24	107.109	4.463
9	20.011	608.909	22	119.861	5.448
10	26.413	600.082	26	128.541	4.944
	Total:		241	1139.875	

Total length including acknowledgments = 1541.948

All cues are cleared at 2932.98 [s]

Figure 13. Supplements in <input-summary> for Experiment No. 3.

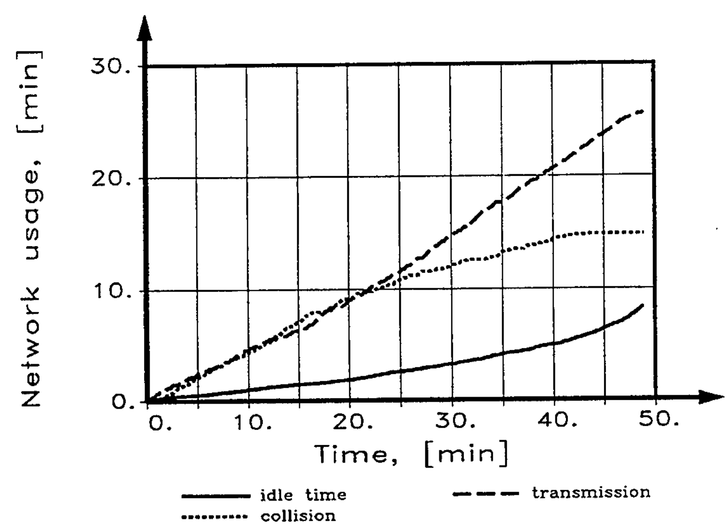

Figure 14. Cumulative network usage times in Experiment No. 3.

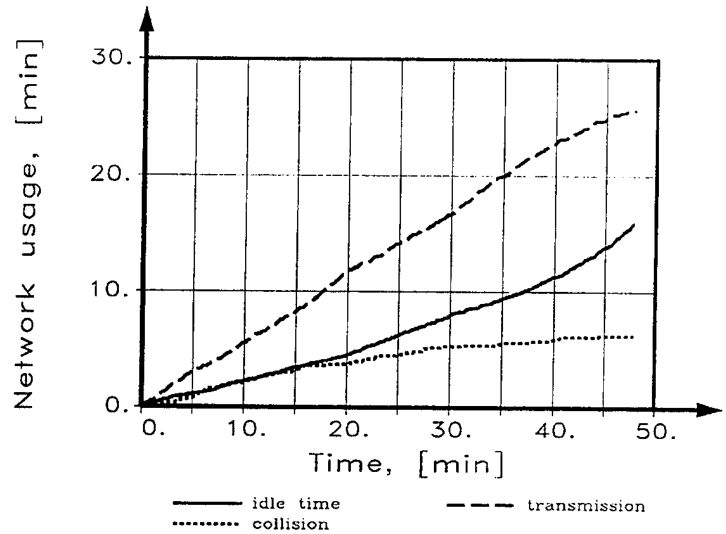

Figure 15. Cumulative network usage times in Experiment No. 4.

6. SUMMARY AND CONCLUSIONS

A battlefield communications network that consists of a limited number of independent nodes that all broadcast on the same radio channel has been considered. Control of access to the channel is important when several nodes try to broadcast at the same time. A possible way to control the access is to install at each node a controller that regulates the node's access in such a way that the overall throughput rate of information is increased. This report describes a computer model of such a network. The purpose of the model is to test experimentally algorithms for the distributed network controllers.

The computer code models the network at a high level of abstraction. This makes the model simple and reduces computing time. An experiment with BATNET typically requires 0.1 to 0.5 percent of the real time of the modeled event. That is, one hour of heavy traffic in a battlefield network can be simulated by BATNET in a fraction of a minute on a Cray Y-MP computer. The simple structure of the model facilitates code modifications to accommodate new hardware concepts and new communication protocols.

Preliminary experiments with the model indicate that the network throughput characteristics are quite sensitive to network access parameters. Therefore, an enhancement of information throughput can likely be achieved by a dynamic adjustment of these parameters. The model BATNET *will be used for experimental testing and fine-tuning of dynamic control rules.*

7. REFERENCE

Virginia A. T. Kaste, Ann E. M. Brodeen, and Barbara D. Broome, An Experiment to Examine Protocol Performance Over Combat Net Radios, Ballistic Research Laboratory Memorandum Report BRL-MR-3978, Aberdeen Proving Ground, MD 21005, June 1992.

INTENTIONALLY LEFT BLANK

Appendix.

LIST OF THE PROGRAM BATNET

INTENTIONALLY LEFT BLANK

```
      program batnet
*
*   Battlefield net model to simulate the behavior of shared-channel
*   multi-node net.
*   Aivars Celmins fecit 14 March 1994.  Version 12 September 1994.
*
      implicit none
      real flcue(10,5,301),ttrans(10),tculen(10),wculen(10),freqms(10)
     A ,tmslen(10),delmax(10),delave(10),delpar(3),timeli(2,501)
     B ,stat(3,5),restat(3,5),tclear
     C ,tmmin,tackno,dtimon,tend,tinter,thold,tinit
      integer lastms(10),incue(10,4,301),ntrans(10),nollen(10)
     A ,nculen(10),ntimel(501),nostor
     B ,nadim,nfdim,nidim,ncdim,ntldim,nrnods,ka,iseed
      character head*70,ida*8,icl*8
*
      parameter(nadim=10,nfdim=5,nidim=4,ncdim=301,ntldim=501)
* Dimensions of various arrays
      parameter(tmmin=0.627)
      parameter(tackno=0.787)
* tmmin  is the minimum length of a message, [s]
* tackno  is the length of message acknowledgment,  [s]
*
      call reader(head,nrnods,freqms,tmslen,nadim,iseed,tend
     A ,tinter,thold,delpar,dtimon)
* Reader has provided the following network parameters:
* nrnods = number of nodes in the net
* freqms(10) = frequency of message for each node,  [1/s]
* tmslen(10) = medium message length for each node, [s]
* iseed  =  seed number for the random number generator
* tend   = end time of message generation [s].  (Start time is zero.)
* tinter  = the minimum separation of messages to avoid interference, [s].
* thold  = hold time for waiting for acknowledgment, [s]
* delpar(3) = three parameters of the delay function used in <nodupd>
*        (1) = factor in the delay function (delay time interval) [s],
*        (2) and (3) - presently not used
* dtimon = time interval [s] for monitoring of net activities by  <monitor>
*
* Next read the day and time from fort.20 where the script has placed them
      read(unit=20,'(a8)') ida,icl
*
      call writer(ida,icl,head,nrnods,freqms,tmslen,nadim,iseed
     A ,tend,tinter,thold,delpar,dtimon)
* This makes a print file fort.37 = <input-summary> with
*  comprehensive input summary
*
      tinit=0.
      do 26 ka=1,nrnods
      call nodini(nrnods,ka,tmmin,tinit,freqms,tmslen,tend
     A ,lastms,flcue,incue,nadim,nfdim,nidim,ncdim,iseed)
* This generates message queues and stores the queue information in
* lastms, flcue, and incue.
* lastms(ka) = identification number of last message at node <ka>
```

```
* flcue = floating point information about the queues
* incue = integer information about the queues
*        See <nodini> for detailed description.
*
    if(flcue(ka,1,lastms(ka)).lt.tend) then
    write(37,21)' Stop because nodini has not sufficient storage'
   A ,' to initiate queues',' Last message at node ',ka
   B ,' is submitted at ',flcue(ka,1,lastms(ka))
   C ,' End time is ',tend,' Storage size is ',ncdim
   D ,', index of last stored message is ',lastms(ka)
  21 format(/,a,a,/,a,i2,a,1pe12.5,/,a,1pe12.5,/,a,i5,a,i5)
    close(unit=37)
    stop
    endif
*
  26 continue
*
    call write2(tinit,nrnods,lastms,tackno,thold
   A ,flcue,incue,nadim,nfdim,nidim,ncdim)
* This supplements the input information on the file fort.37=<input-summary>
* with some statistics about the queues prepared by <nodini>.
*
    nostor=0
* If nostor.eq.0 then store results in files for plotting and examination
* Next carry out the network experiment for this case.
    call excase(nostor,tclear
   A ,ida,icl,head,nrnods,iseed,tend,tinter,thold,delpar,dtimon
   B ,tmmin,tackno,tinit,lastms,flcue,incue
   C ,ttrans,ntrans,nollen,nculen,tculen,wculen,delmax,delave
   D ,timeli,ntimel,stat,restat
   E ,nadim,nfdim,nidim,ncdim,ntldim)
*
    write(37,'(/,a,0pf10.2,a)') ' All queues are cleared at'
   A                             ,tclear,' [s]'
    close(unit=37)
*
    stop
    end
*
****************
*
    subroutine reader(head,nrnods,freqms,tmslen,nadim,iseed,tend
   A ,tinter,thold,delpar,dtimon)
*
* To read data from fort.21 for network traffic simulation.
* Aivars Celmins fecit 15 March 1994. Version 20 May 1994.
*
    implicit none
    real freqms(nadim),tmslen(nadim),tend,tinter,thold,delpar(3)
   A ,din(2),dtimon
    integer nrnods,nadim,ka,j,nin,kb,iseed
    character head*70, dum*1
*
    head=' Default input. 20 May 1994. Delay interval 15 [s]'
    nrnods=4
    iseed=0
```

```
      tend=600.
      tinter=0.5
      thold=1.0
      delpar(1)=15.
      delpar(2)=1.
      delpar(3)=1.
      do 5 ka=1,nadim
      freqms(ka)=0.05
      tmslen(ka)=5.
    5 continue
* The default values will be used if the input file fort.21 is empty
* or incomplete.
*
      rewind(unit=21)
*
      read(21,11,end=21,err=21) head
   11 format(a70)
      read(21,12,end=21,err=21) dum
   12 format(a1)
      read(21,*) tinter,thold
* tinter = minimum interval between messages to avoid collision, [s]
* thold  = hold time, [s], to wait for acknowlwedgement
      read(21,12,end=21,err=21) dum
      read(21,*) (delpar(j),j=1,3)
* delpar = message delay control parameters
*
      read(21,12,end=21,err=21) dum
      read(21,*,end=21,err=21) nin
      nrnods=min(nadim,nin)
*  The number of nodes in the network <nrnods> cannot be larger than <nadim>
*
      read(21,12,end=21,err=21) dum
      do 17 ka=1,nin
      read(21,*,end=21,err=21) kb, din(1),din(2)
        if(kb.le.nadim) then
*  The maximum number of nodes that can be stored is <nadim>.
        freqms(kb)=din(1)
        tmslen(kb)=din(2)
        endif
   17 continue
*  freqms = frequency of messages, [1/s]
*  tmslen = medium message length, [s]
*
      read(21,12,end=21,err=21) dum
      read(21,*,end=21,err=21) iseed
*  iseed = seed for the random number function <unif>
*
      read(21,12,end=21,err=21) dum
      read(21,*,end=21,err=21) tend
*  tend = end time of message generation
*
      read(21,12,end=21,err=21) dum
      read(21,*,end=21,err=21) dtimon
*  dtimon = monitoring time used by the subroutine monitor
*
   21 continue
```

```
      close(unit=21)
      return
      end
*
*****************
*
      subroutine writer(ida,icl,head,nrnods,freqms,tmslen,nadim,iseed
     A ,tend,tinter,thold,delpar,dtimon)
*
* This makes a file <input-summary> with a comprehensive input summary
* Aivars Celmins fecit 16 March 1994.   Version 20 May 1994
*
      implicit none
      real freqms(nadim),tmslen(nadim),tend,tinter,thold,delpar(3)
     A ,dtimon
      integer nrnods,nadim,ka,j,iseed
      character head*70,ida*8,icl*8
*
      open(unit=37,file='input-summary')
      rewind(unit=37)
*
      write(37,'(a70)') head
      write(37,*)' Collision interval [s], hold time [s]'
      write(37,10) tinter,thold
   10 format(2x,2(0pf10.4))
      write(37,*) ' Message delay parameters (3)'
      write(37,12) (delpar(j),j=1,3)
   12 format(2x,3(1x,0pf10.2))
      write(37, *) ' Number of nodes in the network'
      write(37, '(5x,i10)') nrnods
*
      write(37,*) ' Node No.  Messg. freq. [Hz]  Messg. length [s]'
      do 17 ka=1,nrnods
      write(37,13) ka,freqms(ka),tmslen(ka)
   13 format(3x,i3,10x,0pf7.3,10x,0pf7.1)
   17 continue
*
      write(37,*) ' Seed for the random number generator'
      write(37,'(5x,i10)') iseed
*
      write(37,*) ' End time of message generation [s]'
      write(37,21) tend
   21 format(3x,0pf10.1)
*
      write(37,*) ' Monitoring time [s]'
      write(37,21) dtimon
*
      write(37,11) ida, icl
   11 format(/'Computing date ',a8,'.   Computing time ',a8,'.')
*
*     close(unit=37)
      return
      end
*
*****************
*
```

```
      subroutine write2(timup,nrnods,lastms,tackno,thold
     A ,flcue,incue,nadim,nfdim,nidim,ncdim)
*
* This supplements the file < input-summary > with message statistics
* Aivars Celmins fecit 20 May 1994.
*
      implicit none
      real timup,tackno,thold,flcue(nadim,nfdim,ncdim),tsubm,tsum
     A ,avleng,tstart,total,actot
      integer nrnods,lastms(nadim),incue(nadim,nidim,ncdim)
     A ,nadim,nfdim,nidim,ncdim,numtot,ka,kb,number
*
      write(37,10) timup
   10 format(/'    Summary of unsent messages at time=',0pf7.3/)
      write(37,12)
   12 format('    First subm.  Last subm.  Total msg.  Total msg.'
     A ,'  Average msg.'
     B ,//,  ' Node time [s]    time [s]    number    length [s].'
     C ,'  length [s]'/)
      total=0.
      numtot=0
      actot=0.
*
      do 31 ka=1,nrnods
      tsubm=0.
      number=0.
      tsum=0.
      avleng=0.
        if(lastms(ka).gt.0) then
*
        tstart=flcue(ka,1,1)
        do 21 kb=1,lastms(ka)
        if(incue(ka,2,kb).eq.1) goto 21
        number=number+1
        tsum=tsum+flcue(ka,2,kb)
        actot=actot+flcue(ka,2,kb)
        if(incue(ka,1,kb).ne.0) actot=actot+thold+tackno
* To obtain the total length including acknowledgment, add hold time and
* acknowledgment length to queued message length if addresse is not "world"
        tsubm=flcue(ka,1,kb)
   21   continue
*
        total=total+tsum
        numtot=numtot+number
        if(number.gt.0) avleng=tsum/float(number)
        endif
      write(37,15) ka,tstart,tsubm,number,tsum,avleng
   15 format(i3,1x,0pf10.3,4x,0pf10.3,6x,i3,6x,0pf10.3,3x,0pf10.3)
   31 continue
      write(37,39) numtot,total
   39 format(/20x,'Total:     ',i4,6x,0pf10.3)
      write(37,41) actot
   41 format(/2x,'Total length including acknowledgments =',0pf10.3)
*
      return
      end
```

```
*
*************************
*
      subroutine excase(nostor,tclear
     A ,ida,icl,head,nrnods,iseed,tend,tinter,thold,delpar,dtimon
     B ,tmmin,tackno,tinit,lastms,flcue,incue
     C ,ttrans,ntrans,nollen,nculen,tculen,wculen,delmax,delave
     D ,timeli,ntimel,stat,restat
     E ,nadim,nfdim,nidim,ncdim,ntldim)
*
* This routine runs the battle network until all messages are sent.
* Fecit 19 August 1994
*
* nostor =0: store statistics in files for plotting, =1: do not store
* tclear =  time at which all message queues have been cleared.
*
* ida,icl = date and time of computation
* head = alphanumeric identification of case to be run
* nrnods = number of nodes
* iseed = seed number for random number generator
* tend = end time of message generation [s]
* tinter = interference interval alpha [s]
* thold = hold time while waiting for acknowledgment [s]
* delpar(3) = three parameters for access control
* dtimon = monitoring time for calculaitons of averages [s]
*
* tmmin  = 0.627 is the minimum length of a message, [s]
* tackno = 0.787 is the length of message acknowledgment,  [s]
* tinit = initial time of network experiment [s]
* lastms = identification number of last message in each queu
* flcue  =  floating point information about the queues (see subr. nodini)
* incue  =  integer information about the queues  (see subr. nodidi)
*
* ttrans =  transmission (send) time for the top message in the queue
* ntrans =  number (ID) of the top message (first in line to be sent)
* nollen =  number of previous unsuccessfully transmitted messages
* nculen =  queue length (number of messages waiting)
* tculen =  queue length in [s] (sum of the lengths of waiting messages)
* wculen =  weighted length of each que queue.
* delmax =  largest time delay (waiting time) in each queue [s]
* delave =  average time delay (waiting time) in each queue [s]
*
* timeli =  time-line storage: time & time interval [s]
* ntimel =  activity code for the time interval specified by timeli(..,2)
* stat   =  statistics of net usage, see subroutine <monitor>
* restat =  reference statistics (weighted averagings of stat)
*
* nadim,nfdim,nidim,nodim,ntldim = dimensions of the various arrays.
*
* The following code is for the time line and the overall net activity.
* netuse  =  net usage of time intervals (stored in <ntimel> and
*             written in the file <time-line> )
*           1 - not used (idle time)
*           2 - successful and acknowledged transmission
*           3 - colliding message time
*           4 - lost time due to interference
```

```
*       5 - message sent but not acknowledged (receiver failure)
*
*
      implicit none
      real tclear,flcue(nadim,nfdim,ncdim),ttrans(nadim)
     A ,tculen(nadim),wculen(nadim),delmax(nadim),delave(nadim)
     B ,delpar(3),timeli(2,ntldim),stat(3,5),restat(3,5)
      integer nostor,lastms(nadim),incue(nadim,nidim,ncdim)
     A ,ntrans(nadim),nollen(nadim),nculen(nadim),ntimel(ntldim)
      real tmmin,tackno,dtimon,tend,tinter,thold,tinit,timup,tisent
     A ,timsnd,dumtim,ttend,timesg,tlmesg,avlent
      integer nadim,nfdim,nidim,ncdim,ntldim,nrnods,ka,messnd
     A ,nrmesg,nodsnd,netuse,interf,nrtime,iseed
      character head*70,ida*8,icl*8
*
*     parameter(nadim=10,nfdim=5,nidim=4,ncdim=301,ntldim=501)
*
      if(nostor.eq.0) call openfil(ida,icl,head,dtimon,nrnods)
* This opens and starts files for the storage of various net statistics
      nrtime=0
      timup=tinit
* Next statement starts loop by sending messages and monitoring traffic.
* The end of the loop is before statement 97.
   37 messnd=0
*
* Next call the control routine to compute the parameters <delpar> of
* the delay function that is used in <nodupd>.
* The changes of <delpar> depend on the results <stat> and <restat>
* obtained by the monitoring soubroutine <monitor>
* The current values of the delay parameters <delpar> are written by
* the routine <control> into unit 48 = <control-line>
*
      call control(nrtime,timup,delpar,dtimon
     A ,avlent,stat,restat,nostor)
*    B ,moante,mocons,axante,axcons,naxdim)
* To control the parameters of the access delay function
*
      tinit=timup
      timsnd=max(tinit,tend)*1000.
      do 54 ka=1,nrnods
      call nodupd(ka,timup,lastms,nrmesg,timesg,tlmesg,tisent
     A ,nollen,nculen,wculen,tculen
     B ,flcue,incue,nadim,nfdim,nidim,ncdim,delpar,iseed)
* This finds the transmission time of the first message in each queue.
      ntrans(ka)=nrmesg
      ttrans(ka)=tisent
      if(nrmesg.eq.0) goto 54
* Next find the smallest send time =  timsnd [s]  among active nodes
        if(tisent.lt.timsnd) then
        nodsnd=ka
        timsnd=tisent
        messnd=nrmesg
        endif
   54 continue
*  Now <nodsnd> is the transmitting node (first in line), and <messnd>
*  is the ID-number of the message to be transmitted.
```

```
    if(messnd.eq.0) goto 97
*  Branch to stop if all queues are empty
*
    if(nostor.eq.0) then
    call strcue(timup,nculen,nollen,wculen,tculen,nrnods,nadim)
*  Write information about queue lengths in three <cues-**> files showing
*  nculen, nollen, wculen, and tculen  for all nodes
*
    call wdelay(nrnods,timup,lastms,delmax,delave
   A ,flcue,incue,nadim,nfdim,nidim,ncdim)
*  Compute and write delay (waiting) times for each node in file <cues-de>
*  delmax = largest time delay (waiting time) in each queue [s]
*  delave = average time delay (waiting time) in each queue [s]
    endif
*
*  Write time usage information in <time-line>.
    netuse=1
    if(nostor.eq.0) write(41,63) timsnd,timsnd-tinit,netuse
  63 format(2(2x,0pf10.4),3x,i3)
*  Network was not used in the last interval up to <timsnd>
    nrtime=nrtime+1
    timeli(1,nrtime)=timsnd
    timeli(2,nrtime)=timsnd-tinit
    ntimel(nrtime)=netuse
*
    call monitor(nrtime,timeli,ntimel,ntldim,dtimon,stat,
   A restat,avlent,nostor)
*  This routine computes various network activity statistics and writes
*  the results in fort.43=<monitor-line> and fort.44=<monitor2-numbers>
*
    if(nrtime.ge.ntldim-1)
   A call reduce(nrtime,timeli,ntimel,ntldim,dtimon)
*  If array <timeli> overflows then remove old information from it.
*
*  Next advance time to the next message sent
    timup=timsnd+flcue(nodsnd,2,messnd)
* Add message length to reference time
    if(incue(nodsnd,1,messnd).ne.0) timup=timup+thold+tackno
* If message was not addressed to the world then add also hold time
* and acknowledgment time
    incue(nodsnd,2,messnd)=incue(nodsnd,2,messnd)+1
*  Indicate that message has been transmitted one more time
    netuse=2
*  Net-usage code for successful transmission in this interval
    incue(nodsnd,3,messnd)=1
    flcue(nodsnd,5,messnd)=timsnd
*  Indicate that message has been sent and acknowledged.
*
*  Next check if this message suffers interference from any other messages
    ttend=timsnd+flcue(nodsnd,2,messnd)
*  End time of the message without acknowledgment
    interf=0
*  Interference indicator.
    do 73 ka=1,nrnods
    if(ka.eq.nodsnd.or.nculen(ka).eq.0) goto 73
*  nculen(ka) is zero if the queue of node <ka> is empty
```

```
      if(timsnd+tinter.le.ttrans(ka)) goto 73
* Branch if no interference from the node  <ka>
      interf=1
      incue(nodsnd,3,messnd)=3
* Indicate collision for the first broadcasting node
      incue(ka,2,ntrans(ka))=incue(ka,2,ntrans(ka))+1
      incue(ka,3,ntrans(ka))=3
      flcue(ka,5,ntrans(ka))=ttrans(ka)
* Indicate send attempt and collision for the other node (the collider)
      dumtim=ttrans(ka)+flcue(ka,2,ntrans(ka))
      ttend=max(ttend,dumtim)
* Reference time is the maximum end time of all collided messages without
* acknowledgments.
   73 continue
         if(interf.ne.0) then
         timup=ttend
         netuse=3
* Indicate that net was used by colliding messages in the time interval
* ending with <ttend>
         endif
*
**************
*****  Future: In case of no collison (interf=0) check here for corruption
*****  by noise and whether the receiving node was operational.
**************
*
* Write time-line information in the file  fort.41=<time-line>  :
      if(nostor.eq.0) write(41,63) timup,timup-timsnd,netuse
* Next store the end point of this time interval and the activity code
* in the time-line arrays <timeli> and <ntimel>, respectively.
      nrtime=nrtime+1
      timeli(1,nrtime)=timup
      timeli(2,nrtime)=timup-timsnd
      ntimel(nrtime)=netuse
*
      call monitor(nrtime,timeli,ntimel,ntldim,dtimon,stat,
     A restat,avlent,nostor)
* This routine computes network activity statistics and writes network
* time usages in fort.43 = <monitor-line> and the nubers of network
* accesses in fort.44 = <monitor2-numbers>
*
* Now have broadcast a message and advanced the reference time to the
* end of that message (including colliding messages, if any).
      if(timup.lt.tend*1000.) goto 37
* Branch to 37 for the broadcasting of the next message.
* The conditional branch is to guard against infinite looping.
* <tend> is the end time of *message generation* and the queues should
* be empty long before <tend*1000>.
*
   97 continue
      tclear=timup
*
      if(nostor.eq.0) call closfil
* Close the files with statistical information (see openfil)
*
      return
```

```
    end
*
*****************
*
    subroutine nodini(nrnods,node,tmmin,tinit,freqms,tmslen,tend
   A ,lastms,flcue,incue,nadim,nfdim,nidim,ncdim,iseed)
*
* This generates a list of future messages at the node No. <node>.
* Aivars Celmins fecit 4 March 1994.  Version 17 March 1994
*
* nrnods = number of nodes in the net (needed for addressee determination)
* node  = identification number of the node
* tmmin = minimum length of a message (message head), [s]
* tinit = initial time, [s]
* freqms(nadim) = frequency of messages for each node,  [messages / s]
* tmslen(nadim) = medium lenght of messages for each node, [s]
* tend  = end time for message generation, [s]
*
* lastms(nadim)  =  number of last message in each list
* flcue(nadim,nfdim,ncdim) = floating number items in the list
*    First index = node number (= "node")
*    Second index: 1 = submission time [s]
*              2 = message length, [s]
*              3 = priority in the range [0,10]
*              4 = weight, computed in subroutine <nodupd>
*              5 = last transmission time, [s]
*    Third index = number of the message (up to ncdim messages in the list)
* incue(nadim,nidim,ncdim) = integer number items in the list
*    First index = node number (= "node")
*    Second index: 1 = identification number of addresee (0 means "world")
*                2 = number of tried transmissions
*                3 = acknowledgment (0 - not sent, 1 - acknowledged,
*                            2 - not acknowl., 3 - collided)
*              4 = activity indicator (0 - active, 1 - dormant)
*
*    Third index = number of the message
* nadim,nfdim,nidim,ncdim  =  dimensions of various arguments
* iseed  =  seed number for the random number generator.
*
    implicit none
    real tinit,freqms(nadim),tmslen(nadim),tend
   A ,flcue(nadim,nfdim,ncdim)
    integer nrnods,node,lastms(nadim),incue(nadim,nidim,ncdim)
   A ,nadim,nfdim,nidim,ncdim
    real tmmin,tsubm,unif,delt
    integer nrmes,ka,kb,nstore,ntest,nadr,iseed
*
    if(lastms(node).gt.0.and.flcue(node,1,lastms(node)).ge.tend)
   A goto 144
* Branch to return if the last message in the queue of <node> has a
* submission time larger than <tend>.
    tsubm=tinit
      if(tinit.le.0.) then
* The following initialization if initial time <tinit> is zero.
* (Normal usage of this routine.)
      nrmes=0
```

```
      lastms(node)=nrmes
      do 19 ka=1,ncdim
      do 15 kb=1,nfdim
      flcue(node,kb,ka)=0.
  15   continue
      do 17 kb=1,nidim
      incue(node,kb,ka)=0
  17   continue
  19   continue
      goto 111
      endif
* Branch to 111 and generate new messages in the queue if this is the
* first call.  Else remove all sent messages before adding new ones.
*
    nstore=0
    ntest=1
  24 if(flcue(node,2,ntest).le.0.) then
      nrmes=nstore
      if(nrmes.gt.0) tsubm=flcue(node,1,nrmes)
      goto 111
* Branch to 111 because there are no more old messages in the queue.
    endif
     if(incue(node,3,ntest).eq.1) then
* If message has been sent and acknowledged (incue( ,3, )=1) then
* drop the message from the queue and test the next message.
      ntest=ntest+1
      if(ntest.gt.ncdim) goto 111
      goto 24
      else
* Store the message <ntest> in the queue since it has not been acknowledged.
      nstore=nstore+1
      nrmes=nstore
      tsubm=flcue(node,1,nrmes)
      do 33 ka=1,nfdim
      flcue(node,ka,nstore)=flcue(node,ka,ntest)
  33   continue
      do 35 ka=1,nidim
      incue(node,ka,nstore)=incue(node,ka,ntest)
  35   continue
     endif
      ntest=ntest+1
      if(nstore.ge.ncdim.or.ntest.ge.ncdim) goto 111
      goto 24
*
 111 lastms(node)=nrmes
* Enter here and make a queue of new messages
    nrmes=nrmes+1
    if(nrmes.gt.ncdim) goto 144
* Branch to return: queue full.
    lastms(node)=nrmes
* Else find the time increment to the next message.
    delt=unif(iseed)*(2./freqms(node))
    tsubm=tsubm+delt
    flcue(node,1,nrmes)=tsubm
* Submission time, [s]
    flcue(node,2,nrmes)=tmmin+unif(iseed)*2.*tmslen(node)
```

```
* Message length [s] is at least tmmin [s].
    flcue(node,3,nrmes)=float( int( 10.*unif(iseed) ) )
* Priority in the range [0.0, 10.0]
    flcue(node,4,nrmes)=0.
    flcue(node,5,nrmes)=0.
* Next determine addressee.  Presently send either to world (nadr=0)
* or to a single node with a number different from "node"
    nadr=int(unif(iseed)*float(nrnods+1)*0.999)
    if(nadr.eq.node) nadr=nadr-1
    incue(node,1,nrmes)=nadr
    do 122 ka=2,nidim
    incue(node,ka,nrmes)=0.
 122 continue
*
    if (tsubm.lt.tend) goto 111
* Add new messages until the last one is  .ge.  tend
*
    if(nrmes.lt.ncdim) then
       do 139 ka=nrmes+1,ncdim
       flcue(node,2,ka)=0.
 139   continue
    endif
*
 144 return
    end
*
******************
*
    subroutine nodupd(node,timup,lastms,nrmesg,timesg,tlmesg,tisent
    A ,nollen,nculen,wculen,tculen
    B ,flcue,incue,nadim,nfdim,nidim,ncdim,delpar,iseed)
*
* Subroutine updates the queue at the node No. <node> by checking priorities
* of submitted messages and indicating on return which message will be
* broadcast and at what time.
* Aivars Celmins fecit 10 March 1994.  Version 22 March 1994.
*
* node = number of the node
* timup =  reference for update time (consider all messages submitted
*        before this time or first message submitted after timup
*        if there are none before timup.)
* lastms =  last message in the queue of <node>
** The ** items are provided by this routine
** nrmesg  =  identification number of the message that is selected
** timesg  =  time at which the selected message was submitted [s]
** tlmesg  =  length of the selected message [s]
** tisent  =  time at which the selected message will be sent [s]
**                (if net is free at that time)
** nollen  =  number of previous unsuccessfully transmitted messages
** nculen  =  present length of queue (number of messages in the queue)
** wculen  =  length of the queue in terms of message weights
** tculen  =  length of the queue in terms of message time [s]
*
*  flcue, incue  = floating and integer message queue arrays
*              See <nodini> for a description.
*  nadim, nfdim, nidim, ncdim = dimensions of flcue and incue, see below
```

```
*  delpar = network access delay function parameters
*  iseed  = seed number for teh random number generator
*
    implicit none
    real timup,timesg,tlmesg,tisent,wculen(nadim),tculen(nadim)
   A ,flcue(nadim,nfdim,ncdim),delpar(3)
    integer node,lastms(nadim),nrmesg,nollen(nadim),nculen(nadim)
   A ,incue(nadim,nidim,ncdim),nadim,nfdim,nidim ,ncdim
    real wefu,pri,delt,defu,unif,wmax,timarg,weight
    integer nur,ncue,ka,kb,nrep,kmax,iseed
*
    wefu(pri,nur,delt)=
   A (1.+pri+float(nur)) * max( 0.01, exp(-((delt-600.)/600.)**2) )
* Weight is a function of priority, number of submissions, and time since
* submission.  The maximun is at 10 minutes, the lower bound at 31 minutes
* after submission,
    defu(pri)=delpar(1)*(1.0-0.09*pri)*unif(iseed)
* Delay function computes delay in [s] for priority pri
*
* First check if there are messages submitted before timup
    ncue=0
    nollen(node)=0
* This counts the active messages in the queue.
    do 23 ka=1,lastms(node)
    kb=ka
       if(incue(node,3,ka).eq.1) then
       flcue(node,4,ka)=0.
       goto 23
* Set weight=0 and branch if this message has been acknowledged.
       endif
    if(flcue(node,1,ka).gt.timup) goto 39
* Branch when message was submitted after timup
    ncue=ncue+1
* <ncue>  counts messages that were submitted at or before <timup> and
* have not been successfully transmitted yet.
    if(incue(node,2,ka).gt.0) nollen(node)=nollen(node)+1
* <nollen>  counts messages that have been transmitted unsuccessfully
  23 continue
* Now either ncue=0 which means that all messages have been sent
* and no further messages will be broadcast from this node,
* or  ncue > 0 in  which case there are old messages waiting
*
      if(ncue.eq.0) then
      nrmesg=0
      timesg=0.
      tlmesg=0.
      tisent=0.
      nculen(node)=0
      wculen(node)=0.
      tculen(node)=0.
      return
      endif
*
* The following is entered either from inside the loop 23 or
*          with ncue > 0 after completion of the loop
  39 if(ncue.eq.0) then
```

```
* In this case the queue is empty but there will be a message submitted
* after the reference time <timup>
     nrmesg=kb
     timesg=flcue(node,1,kb)
     tlmesg=flcue(node,2,kb)
     tisent=timesg
* The next message that will be submitted to the queue at
* tisent=timesg > timup  can be broadcast at <timesg> without delay
* because the net is already free.
     nculen(node)=0
     wculen(node)=0.
     tculen(node)=0.
* Nothing in the queue at the present reference time <= timup
     return
     endif
*
* In the remaining cases have  ncue >= 1  messages in the queue.
* Compute message weights and select the message with the highest weight
     wmax=0.
     tculen(node)=0.
     wculen(node)=0.
     nculen(node)=0
     do 45 ka=1,kb
     if(flcue(node,2,ka).le.0..or.incue(node,3,ka).eq.1) goto 45
* Branch if message has zero length or has been acknowledged
     if(flcue(node,1,ka).gt.timup) goto 45
* Branch if  message  is a future message
     nrep=incue(node,2,ka)
     timarg=timup-flcue(node,1,ka)
     weight=wefu(flcue(node,3,ka),nrep,timarg)
     flcue(node,4,ka)=weight
          if(weight.gt.wmax) then
          wmax=weight
          kmax=ka
* Find the message with the largest weight
          endif
     tculen(node)=tculen(node)+flcue(node,2,ka)
     wculen(node)=wculen(node)+weight
     nculen(node)=nculen(node)+1
  45 continue
*
     nrmesg=kmax
     timesg=flcue(node,1,kmax)
     tlmesg=flcue(node,2,kmax)
     tisent=timup+defu(flcue(node,3,kmax))
*
     return
     end
*
******************
*
     subroutine wdelay(nrnods,timup,lastms,delmax,delave
    A ,flcue,incue,nadim,nfdim,nidim,ncdim)
*
* Compute and write delay times (waiting times) in fort.34=<cues-de>
* Aivars Celmins fecit  18 March 1994
```

```
*
* The routine computes the following:
* delmax = largest time delay (waiting time) in each queue [s]
* delave = average time delay (waiting time) in each queue [s]
*
    implicit none
    real timup,delmax(nadim),delave(nadim),flcue(nadim,nfdim,ncdim)
    integer nrnods,lastms(nadim),incue(nadim,nidim,ncdim)
   A ,nadim,nfdim,nidim,ncdim
    real timsum,dtim
    integer ka,kt,kb,j
*
* Next find delay times in each node
    do 34 ka=1,nrnods
    delmax(ka)=0.
    delave(ka)=0.
    timsum=0.
    kt=0
*
    do 25 kb=1,lastms(ka)
    if(flcue(ka,1,kb).ge.timup) goto 29
    if(incue(ka,3,kb).eq.1) goto 25
* Branch if message has been sent
    dtim=timup-flcue(ka,1,kb)
    timsum=timsum+dtim
    kt=kt+1
    delmax(ka)=max(delmax(ka),dtim)
  25 continue
*
  29 delave(ka)=timsum/float(max(1,kt))
  34 continue
*
* Write results in <cues-de>
    write(34,44) timup,(delmax(j),delave(j),j=1,nrnods)
  44 format(0pf8.2,(4(1x,1pf8.1,1pf8.1)))
*
    return
    end
*
****************
*
    subroutine monitor(nrtime,timeli,ntimel,ntldim,dtimon,stat,
   A restat,avlent,nostor)
*
* Subroutine to monitor net activities by computing several network
* statisitcs in <stat> and <restat>. The statistics are averages for the
* time interval  <dtimon> [s]  prior to  <timeli(1,nrtime)> [s].
* Aivars Celmins fecit 11 April 1994.  Version 29 July 1994.
*
* nrtime = index of the reference time  timeli(1,nrtime)
* timeli = array representing the time line of network activities
*        timeli(1,nrtime) = reference time [s]
*        timeli(2,nrtime) = interval before the reference time [s]
* ntimel = array indicating the usage of time intervals <timeli(2, ..)>:
*          1 - not used (idle time)
*          2 - successful and acknowledged transmission
```

```
*        3 - colliding message
*        4 - message corrupted by interference (network noise)
*        5 - message sent but not acknowledged (receiver failure)
* ntldim = dimension of the arrays  timeli  and  ntimel
* dtimon = time interval for the averaging [s]
*
* The ** items are provided by this routine
** stat(1,n) = cumulative time [s] in the time interval  dtimon [s],
**          divided by  dtimon.  (Relative length of net usage)
** stat(2,n) = average message length [s] during the time interval dtimon.
** stat(3,n) = number of idle intervals, messages, collisions, etc. in dtimon
*          divided by the total number of accesses (n=2,3,4,5)
*  The index  n  indicates the usage category.   The value of n is
*  retrieved from <ntimel(nrtime)>.
** restat(i,n) =  the same as stat( ) but computed with weighted averages
*           to provide an approximation of previous values
** avlent =  average length [s] of all not colliding messages in  dtimon
* nostor =  if this is .ne.0  then do not write results in files.
*
   implicit none
   real timeli(2,ntldim),dtimon,stat(3,5),avlent,restat(3,5)
   integer nrtime,ntimel(ntldim),ntldim,nostor,j
   real timref,timone,adtone,totmes,cumone,delt,anrmes,anrtot
  A ,recumo,renrms,rewefu,tau,wend,wedelt,redelt,renrto,anrlen
   integer ku,ka
*
   rewefu(tau,wend,wedelt)=wend-(1.-wend)*tau/wedelt
* tau = t - timref
*
* Weight function for the computation of <restat( )> with weighted averaging.
* The weights are greater for past values.   The weighted averages
* approximate preceeding unweighted averages in time.
* The difference  "value - weighted_value" indicates the "trend of the value"
*
   timref=timeli(1,nrtime)
   timone=timref-dtimon
   adtone=max(timone,timeli(1,1)-timeli(2,1))
* The averages will be computed for the time interval ( adtone , timref )
   wend=0.5
* <wend> is the value of the weigth function <rewefu> at time <timref>
   wedelt=min(dtimon,timref-adtone)
* <wedelt> is another parameter of the weight function.  It is determined
* such that <rewefu> equals one at <adtone>  =  adjusted timeone
* and equals  <wend>  at  <timref>
*
   anrtot=0.
   renrto=0.
   totmes=0.
   avlent=0.
   anrlen=0.
   do 107 ku=1,5
* Loop over the five net usage codes <ku>
   anrmes=0.
   stat(2,ku)=0.
*
  -cumone=0.
```

```
    recumo=0.
    renrms=0.
*
    do 46 ka=nrtime,1,-1
    if(timeli(1,ka).le.adtone) goto 91
      if(ntimel(ka).eq.ku) then
        delt=min(timeli(2,ka),timeli(1,ka)-adtone)
        cumone=cumone+delt
          tau=timeli(1,ka)-delt*0.5-timref
        recumo=recumo+delt*rewefu(tau,wend,wedelt)
* This accumulates the net usage time for the usage <ku>
        anrmes=anrmes+1.
          renrms=renrms+rewefu(tau,wend,wedelt)
* Count number of sent messages of type  <ku>
      endif
 46 continue
*
 91 stat(1,ku)=cumone/(timref-adtone)
* Store the relative length of time used for <ku> type messages in dtimon
    tau=(timref+adtone)*0.5-timref
    redelt=(timref-adtone)*rewefu(tau,wend,wedelt)
    restat(1,ku)=recumo/redelt
* The relative length of time by weighted averaging
    if(anrmes.gt.0.) stat(2,ku)=cumone/anrmes
    if(renrms.gt.0.) restat(2,ku)=recumo/renrms
* The average lengths of messages in dtimon.  Unweighted and weighted.
      if(ku.ne.1) then
        anrtot=anrtot+anrmes
        renrto=renrto+renrms
          if(ku.ne.3) then
          anrlen=anrlen+anrmes
* Total number of all not colliding messages sent (colliding message lenght
* can be larger than an individual message; therefore they are not used
* for the computation of the average length).
          totmes=totmes+cumone
* The total length of not-colliding messages in seconds
          endif
      endif
*
     stat(3,ku)=anrmes
     restat(3,ku)=renrms
* Store the number of messages of type <ku>  Unweighted and weighted
*
 107 continue
*  End of loop ku=1,5 over the five network usage modes
     if(anrtot.gt.0.) then
      avlent=totmes/anrlen
      do 127 ku=1,5
      stat(3,ku)=stat(3,ku)/anrtot
      if(renrto.gt.0.) restat(3,ku)=restat(3,ku)/renrto
* Compute the relative number of accesses in each category ku=1,2,3,4,5
* (Divided by the number of accesses, i.e., sum over ku = 2,3,4,5)
 127    continue
     endif
*
    if(nostor.eq.0) then
```

```
      write(43,186) timeli(1,nrtime),avlent,
     A (100.*stat(1,j),stat(2,j),j=1,5),
     B (100.*restat(1,j),restat(2,j),j=1,5)
 186  format(0pf10.3,2x,0pf7.3,5(2x,0pf5.1,2x,0pf6.2)/
     A 19x,5(2x,0pf5.1,2x,0pf6.2))
* Store statistics in <monitor-line>
*
      write(44,187) timeli(1,nrtime),(100.*stat(3,j),j=1,5),
     A (100.*restat(3,j),j=1,5)
 187  format(2x,0pf10.4,5(2x,0pf10.6)/12x,5(2x,0pf10.6))
* Store statistics in <monitor2-numbers>
      endif
*
      return
      end
*
****************
*
      subroutine reduce(nrtime,timeli,ntimel,ntldim,dtimon)
*
* Subroutine to reduce storage in the time line arrays <timeli> and
* <ntimel> by deleting old records and shifting the new records down.
* Messages will not be removed if they are submitted within the monitoring
* time interval dtimon before the last message (to enable the <monitor>
* to calculate network traffic statistics).
* Aivars Celmins fecit 5 April 1994.
*
      implicit none
      real timeli(2,ntldim),dtimon
      integer nrtime,ntimel(ntldim),ntldim
      real endtim,strtim
      integer ka,kb,kc
*
      if(nrtime.le.1) return
      endtim=timeli(1,nrtime)
      strtim=endtim-dtimon
      if(timeli(1,1).ge.strtim) return
* This return because timeli contains only length <dtimon> or less
      do 26 ka=2,nrtime
      if(timeli(1,ka).gt.strtim) goto 36
 26   continue
* In the following case dtimon=0.  Store the nrtime data at position "1"
      timeli(1,1)=timeli(1,nrtime)
      timeli(2,1)=timeli(2,nrtime)
      ntimel(1)=ntimel(nrtime)
      nrtime=1
      return
*
 36   if(ka.eq.2) return
* Return because no record can be deleted
      kc=1
      do 46 kb=ka-1,nrtime
      timeli(1,kc)=timeli(1,kb)
      timeli(2,kc)=timeli(2,kb)
      ntimel(kc)=ntimel(kb)
      kc=kc+1
```

```
 46 continue
    nrtime=kc-1
*
    return
    end
*
********************
*
    real function unif(iseed)
* Obtained from from net on 1 VI 94.
* Here is a uniform 0-1 based on work of L'Ecuyer - Comm. of the ACM,
* Vol 31, number 6 pp. 742-749. He reports detailed, excellent results.
*  Meyer Kotkin kotkin@arl.army.mil
*  USAMSAA Inventory Research Office
*  800 U.S. Customs House
*  Phila. PA 19106 (215)597-8377 DSN:444-3808 FAX:(215)597-2240
* Modified 13 VII 94.  Original in Utilpro/random2.
* Note that this program changes the value of iseed.  Therefore, the
* next call with the same variable provides a different random number
*
    implicit none
    integer ia, iq, ir, iseed, k, m
    parameter (ia=40692,iq=52774,ir=3791,m=2147483399)
*
    k = iseed/iq
    iseed = ia*(iseed - k*iq) - k*ir
    if (iseed.le.0) iseed = iseed + m
    unif = real( dble(iseed)/dble(m) )
    return
    end
*
********************
*
    subroutine openfil(ida,icl,head,dtimon,nrnods)
* Open and start files for the storage of network statistics
*
    implicit none
    real dtimon
    integer nrnods
    character head*70,ida*8,icl*8
*
    open(unit=31,file='cues-nr')
    rewind(unit=31)
    write(31,11) ida, icl
 11 format('Computing date ',a8,'.  Computing time ',a8,'.')
    write(31,'(a70)') head
    write(31,23) dtimon
    write(31,12) nrnods
 12 format(2x,i5,' = number of nodes')
    write(31,13)
 13 format('Time, nrs. of waiting and repeated messages in cues')
*
    open(unit=32,file='cues-we')
    rewind(unit=32)
    write(32,11) ida, icl
    write(32,'(a70)') head
```

```
      write(32,23) dtimon
      write(32,12) nrnods
      write(32,*) 'Time, weight lengths of queues'
*
      open(unit=33,file='cues-ti')
      rewind(unit=33)
      write(33,11) ida, icl
      write(33,'(a70)') head
      write(33,23) dtimon
      write(33,12) nrnods
      write(33,*) 'Time, lengths [s] of information in queues'
*
      open(unit=34,file='cues-de')
      rewind(unit=34)
      write(34,11) ida, icl
      write(34,'(a70)') head
      write(34,23) dtimon
      write(34,12) nrnods
      write(34,*) 'Time, maximum and average delay [s]'
*
      open(unit=41,file='time-line')
      rewind(unit=41)
      write(41,11) ida, icl
      write(41,'(a70)') head
      write(41,*) ' End time [s], interval [s], activity code'
      write(41,*) '  1 = idle, 2 = transmission, 3 = collision'
      write(41,*) '  4 = interference, 5 = not acknowledged'
*
      open(unit=43,file='monitor-line')
      rewind(unit=43)
      write(43,11) ida, icl
      write(43,'(a70)') head
      write(43,23) dtimon
   23 format(1x,0pf10.1,' [s] = the monitored time interval')
      write(43,*)' Column (1) = time; (2)  = active interval length [s]'
      write(43,*) ' (3,4) relative length [%] and length [s] of idling'
      write(43,*) ' (5,6) the same for transmission, (7,8) = collision'
      write(43,*) ' (9,10)  = interference, (11,12) = not acknowledged'
      write(43,*) ' Second line = the same with weighted calculation'
*
      open(unit=44,file='monitor2-numbers')
      rewind(unit=44)
      write(44,11) ida, icl
      write(44,'(a70)') head
      write(44,23) dtimon
      write(44,*) ' Column (1) = time [s];'
      write(44,*)' (2) - (6) = relative numbers [%] of accesses'
     A ,' during dtimon:'
      write(44,*) '  2 = idle, 3 = transmission, 4 = collision'
      write(44,*) '  5 = interference, 6 = not acknowledged'
      write(44,*) ' Second line = the same with weighted calculation'
*
      open(unit=48,file='control-line')
      rewind(unit=48)
      write(48,11) ida, icl
      write(48,'(a70)') head
```

```
      write(48,*) ' Column (1) = time [s]; (2) = first access control ',
     A 'parameter [s];'
      write(48,*) ' (3) - (4) = remaining control parameters'
*
      return
      end
*
***********************
*
      subroutine closfil
* Close the files with statistical information
      close(unit=31)
      close(unit=32)
      close(unit=33)
      close(unit=34)
      close(unit=41)
      close(unit=43)
      close(unit=48)
*
      return
      end
*
***********************
*
      subroutine strcue(timup,nculen,nollen,wculen,tculen,nrnods,nadim)
* Store information about queue lengths in three files showing
* nculen, nollen, wculen, and tculen  for all nodes
* Fecit 18 August 1994
*
      implicit none
      integer nrnods,nadim
      real timup,tculen(nadim),wculen(nadim)
      integer nculen(nadim),nollen(nadim),j
*
      write(31,57) timup, ( nculen(j),nollen(j),j=1,nrnods)
   57 format(0pf10.4,6(4x,i4,1x,i2),(/11x,6(4x,i4,1x,i2)))
*     This is file <cues-nr>
      write(32,58) timup, (wculen(j),j=1,nrnods)
   58 format(0pf10.4,5(3x,1pe10.4),(/11x,5(3x,1pe10.4)))
*     This is file <cues-we>
      write(33,59) timup, (tculen(j),j=1,nrnods)
   59 format(0pf10.4,5(3x,0pf10.4),(/11x,5(3x,0pf10.4)))
*     This is file <cues-ti>
*
      return
      end
*
******************************************
*
      subroutine control(nrtime,timup,delpar,dtimon
     A ,avlent,stat,restat,nostor)
* To control the parameters of the access delay function
* On return  delpar(1) and dtimon might be changed.
* Aivars Celmins fecit 26 July 1994.
*
* nrtime = time step (do not start changing control parameters before
```

```
*       statistics is available at nrtime > 1 )
* timup = current reference time [s]
* The ** items are output:
** delpar(1)  = network access time delay interval [s]
** delpar(2) and delpar(3) = access control parameters for future use
* dtimon  = listening time interval for network monitoring, [s]
*
* avlent = average length [s] of all messages sent during <dtimon>
* stat(1,n) = cumulative time [s] in the time interval  <dtimon> [s],
*         divided by  <dtimon>.  (Relative length of network usage)
* stat(2,n) = average message length [s] during the time interval <dtimon>.
* stat(3,n) = number of of accesses in usage categories divided by
*         the total number of accesses in categories (n=2,3,4,5)
*  the index  n  indicates the usage categories:
*       1 -  not used (idle time)
*       2 -  successful and acknowledged transmission
*       3 -  colliding message
*       4 -  message corrupted by interference (network noise)
*       5 -  message sent but not acknowledged (receiver failure)
* restat(i,n) =  the same as stat( ) but computed with wighted averages
*          to provide an approximation of previous values
* nostor =  if this equals zero then write <delpar> in file <control-line>
*
      implicit none
      real timup,delpar(3),dtimon,avlent,stat(3,5),restat(3,5)
      integer nrtime,nostor,j
**********
***  Future:  Adjust the parameters  <delpar>  taking into account the
***        statistics in <stat> and <restat>.
**********
      if(nostor.eq.0)
     A write(48,'(2x,0pf10.4,4(2x,0pf8.2))') timup,(delpar(j),j=1,3)
     B ,dtimon
* Write the control parameter values in the file unit 48 = <control-line>
* for later examination
*
      return
      end
*
*********************************************
*
```

NO. OF COPIES	ORGANIZATION
2	ADMINISTRATOR ATTN DTIC DDA DEFENSE TECHNICAL INFO CTR CAMERON STATION ALEXANDRIA VA 22304-6145
1	DIRECTOR ATTN AMSRL OP SD TA US ARMY RESEARCH LAB 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR ATTN AMSRL OP SD TL US ARMY RESEARCH LAB 2800 POWDER MILL RD ADELPHI MD 20783-1145
1	DIRECTOR ATTN AMSRL OP SD TP US ARMY RESEARCH LAB 2800 POWDER MILL RD ADELPHI MD 20783-1145
	ABERDEEN PROVING GROUND
5	DIR USARL ATTN AMSRL OP AP L (305)

NO. OF COPIES	ORGANIZATION
3	PM FATDS ATTN AMCPM TF MR FRANK MURPHY MR MICHAEL LYONS MR HENRY SAPHOW BLDG 457 FT MONMOUTH NJ 07703-5027
17	PM FATDS ATTN AMCPM TF A LTC DIXON BLDG 457 FT MONMOUTH NJ 07703-5027
2	DIR US ARMY RESEARCH LAB ATTN AMSRL SS IC MR EMMERMAN MR TOKARCIK 2800 POWDER MILL RD ADELPHI MD 20783-1197
2	DIR US ARMY ASCO ATTN AMSLC AT BLDG 2424 FT MEADE MD 20755-5313
3	CG USMC RSRCH DEV AND ACQ CMD ATTN EXECUTIVE DPM COL STANKOWSKY QUANTICO VA 22134-5000
1	DIR US ARMY RESEARCH OFFICE ATTN SLCRO EL DR GAULT PO BOX 12211 RSRCH TRI PK NC 27709-2211
1	PROJECT MANAGER PM ADCCS ATTN SFAE CC AD MR S ALLEN 4920 UNIVERSITY SQUARE ST 3B HUNTSVILLE AL 35816
2	PROJECT MANAGER PM FAAD C2 ATTN SPIC CC ADCCS FA REDSTONE ARSNL AL 35098-5600
1	PROJECT MANAGER PM ASAS ATTN JTFDO ASAS 1500 PLANNING RESEARCH DR MCLEAN VA 22102
1	PROJECT MANAGER PM ASAS ATTN JTFDO AE C 1500 PLANNING RESEARCH DR MCLEAN VA 22102

NO. OF COPIES	ORGANIZATION
3	PROJECT MANAGER PM CHS ATTN SSAE CC CHS COL OLSON MR LEVINE MR BRYNILDSEN FT MONMOUTH NJ 07703-5000
1	PROJECT MANAGER PM OPTADS ATTN SPIC CC OTDS FT MONMOUTH NJ 07703-5000
3	PEO COMMAND & CONTROL SYSTEMS ATTN SFAE CC MG HARMON MR GIORDANO MR ALBARELLI FT MONMOUTH NJ 07703-5000
1	PEO COMMAND & CONTROL SYSTEMS ATTN SFAE CC SE MR MERCURIO FT MONMOUTH NJ 07703-5000
1	CDR US ARMY CECOM ATTN AMSEL RD AED MR RUTH FT MONMOUTH NJ 07703-5001
3	CDR US ARMY CECOM ATTN AMSEL RD C3 CC DR KLOSE MR QUIGLEY MR STROZYK FT MONMOUTH NJ 07703-5001
1	CDR US ARMY CECOM ATTN AMSEL RD C3 TP E MR GRAFF FT MONMOUTH NJ 07703-5001
1	CDR US ARMY ARDEC ATTN TECH DIR DR DAVIDSON PCTNY ARSNL NJ 07806-5000
1	CDR US ARMY ARDEC ATTN SMCAR FSF MR LEHMAN PCTNY ARSNL NJ 07806-5000
1	CDR US ARMY ARDEC ATTN SMCAR ASH MR LARRY OSTUNI PCTNY ARSNL NJ 07806-5000
1	CDR US ARMY MATERIEL CMD ATTN AMCDRA DR LUCY HAGAN 5001 EISENHOWER AVENUE ALEXANDRIA VA 22333-0001

NO. OF COPIES	ORGANIZATION
2	CMDT US ARMY FIELD ARTILLERY SCHOOL ATTN ATSF TSM C3 COL COOPER FT SILL OK 73503-5000
4	CMDT US ARMY FIELD ARTILLERY SCHOOL ATTN ATSF FSC3 MAJ SELLS CPT WILLIAMS MAJ SCOTT CPT BLAKLEY FT SILL OK 73501
2	CDR USMC RSRCH DEV & ENGRG CMD ATTN PM GROUND C2 QUANTICO VA 22134-5080
1	NATL INST OF STNDRDZTN AND TECH ATTN MR SANDOR SZABO BLDG 220 RM B127 GAITHERSBURG MD 20899
1	DIR SANDIA NATIONAL LABS ATTN DR LARRY CHOATE PO BOX 5800 ALBUQUERQUE NM 87185
1	CDR USACADA ATTN TPIO FT LEAVENWORTH KS 66027-5000
1	HQDA ATTN SARD T N MR HUNTER WOODALL THE PENTAGON ROOM 3E360 WASHINGTON DC 20310-0103
1	HQDA ATTN SAUS OR MR W HOLLIS PENTAGON WASHINGTON DC 20310-0001
2	CMDT US ARMY FIELD ARTILLERY SCHOOL ATTN ATSF CD DIR OF COMBAT DEV FT SILL OK 73503-5000
1	CMDT US ARMY FIELD ARTILLERY SCHOOL ATTN ATSF CD MR DUBLISKY FT SILL OK 73503-5000
1	CMDT US ARMY FIELD ARTILLERY SCHOOL ATTN PRESIDENT FA BD COL ELDER FT SILL OK 73503-5000

NO. OF COPIES	ORGANIZATION
1	DIR ARPA ATTN AST DIR TTO DR JASPER LUPO 1400 WILSON BLVD ARLINGTON VA 22209-2308
6	MAGNAVOX ATTN TOM KLAGE 4 CP MIKE MEIER 10 06 2 CP 1313 PRODUCTION ROAD FT WAYNE IN 46808
3	GE ATCCS SE&I ATTN JOE SLADEWSKI SHEILA HOPKINS RICH STAATS 788 SHREWSBURY AVE TINTON FALLS NJ 07724
2	COMMAND SYSTEMS INC ATTN DEVIN R WILLIS MARK MCCLEARY 1025 GOSHEN ROAD FT WAYNE IN 46808
2	PM SINCGARS ATTN SFAE CM GAR COL DOMINIC F BASILE FT MONMOUTH NJ 07703-5000
1	PM ADDS ATTN SFAE CM ADD COL LELAND H HEWITT FT MONMOUTH NJ 07703-5000
1	PM EPLRS ATTN SFAE CM ADD EPL LTC C FORNECKER FT MONMOUTH NJ 07703-5000
1	PM MSE ATTN SFAE CM MSE COL DAVID GUST FT MONMOUTH NJ 07703-5000
2	CMDT USAFAS ATTN ATSF CCS WALTER W MILLSPAUGH MAJ DRUMMUND FT SILL OK 73503-5600

NO. OF COPIES	ORGANIZATION
4	OPM USAFAS ATTN SFAE ASM FA LARRY YUNG PCTNY ARSNL NJ 07806-5000
1	CDR US ARMY MICOM ATTN AMSMI RD SED MR G CLAYTON REDSTONE ARSNL AL 35898-5260
2	CDR US ARMY TACOM ATTN AMSTA RV MR SARNA MR HALLE WARREN MI 48397-5000
3	CDR USCAC ATTN ATZL CDC COL BAERMAN LTC COOK CPT BROWN FT LEAVENWORTH KS 66027-5300
3	DIR USARL ATTN AMSRL SC IS MR MITCHELL MR RACINE COL BLAKE 115 O'KEEFE BLDG GEORGIA TECH ATLANTA GA 30332-0800

NO. OF COPIES	ORGANIZATION
	ABERDEEN PROVING GROUND, MD
9	DIR USAAMSAA ATTN AMXSY C MR HAL BURKE MR PETE REID MR TOM NOLAN MR PAT WARD MR FRANK FOX MR JOHN DILEO AMXSY G MR JOHN KRAMER AMXSY A MR WALTER CLIFFORD JTCG ME MR ART LAGRANGE
22	DIR USARL ATTN: AMSRL-WT-WF, G HORLEY W DOUSA AMSRL-HR-SA, D TYROL AMSRL-IS-TP, A BRODEEN B BROOME S CHAMBERLAIN B COOPER A DOWNS G HARTWIG R KASTE AMSRL-SC-C, H BREAUX AMSRL-SC-CC, C NIETUBICZ D PRESSEL M TAYLOR C ZOLTANI D HISLEY A CELMINS (3 CP) AMSRL-SC-S, V KASTE AMSRL-SC-SS, M THOMAS AMSRL-SL-C, W HUGHES

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number ARL-MR-244 ____________ Date of Report August 1995 ____________

2. Date Report Received ____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) ____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) ____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. ____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) ____

CURRENT ADDRESS

____
Organization

____
Name

____
Street or P.O. Box No.

____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

____
Organization

____
Name

____
Street or P.O. Box No.

____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)

DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 0001,APG,MD

POSTAGE WILL BE PAID BY ADDRESSEE

DIRECTOR
U.S. ARMY RESEARCH LABORATORY
ATTN: AMSRL-SC-CC
ABERDEEN PROVING GROUND, MD 21005-5067